Amendments to the Specification

Please replace the paragraph at page 4, lines 4 through 8 with the following amended paragraph:

A1

In accordance with one aspect of the invention, a network router comprises queues which store data packets to be forwarded. A scheduler which selects a queue from which a packet is forwarded holds scheduling values, such as [[GBR]] CBR and WFQ counter values, associated with the queues. Scheduling values are compared in a selection network to select the packets to be forwarded.

Please replace the paragraph at page 8, lines 3 through 4 with the following amended paragraph:

A2

~~Figure~~ Figures [[8]] 8A, 8B and 8C ~~illustrates~~ illustrate the hardware for implementing the tree structure in a pipeline architecture.

Please replace the paragraph at page 8, lines 21 through 22 with the following amended paragraph:

A3

~~Figure~~ Figures [[20]] 20A and 20B ~~presents~~ present a hardware implementation of the flip-flop array of Figure 19.

Please replace the paragraph at page 14, line 21 through page 15, line 4 with the following amended paragraph:

A4

At very high line rates, it is not sufficient to complete one incremental tournament every three cycles using the hardware of Figure 4 (or one tournament every 12 cycles for 4096 queues). A higher rate of packet scheduling, one packet per cycle, can be achieved by pipelining the tournament sort as illustrated in ~~Figure 8~~ Figures 8A-C. The circuit of ~~Figure 8~~ Figures 8A-C consists of three pipeline stages, 600, 620, and 640. Each pipeline stage includes all of the logic of the tournament sort engine of Figure 4, but operates only on one level of the tree. The table 200 holding the node states is partitioned across the three pipeline stages. The leaves of the tree, locations 0-7, are held in RAM 610 in pipeline stage 600. The first level of internal nodes, locations 8-11, are held in RAM 630 in pipeline stage 620, and the second level of internal

A4 Cont. nodes, locations 12-13, are held in RAM 650 in pipeline stage 640. Note that there is no need to store the root of the tree, location 14, as the winning node ID and key are output on lines 653 and 654 respectively.

Please replace the paragraph at page 15, line 5 through page 16, line 2 with the following amended paragraph:

Consider, for example, starting with the initial state shown in Figure 1 and changing the counter associated with queue five to 9, as was done in Figure 3. The pipelined circuit of ~~Figure 8~~ Figures 8A-C computes the incremental tournament associated with this change in three cycles. At the start of the first cycle, address register 601 is loaded with the address/ID of the leaf node 5 over line 690. At the same time, the ID portion of compare register 602 is loaded with the same value, and the key portion of compare register 602 is loaded with 9, the new value of the queue's counter over line 691. During the first cycle, 5.9, the new entry for leaf node 5, is written to location 5 in RAM 610 using write address port 606 and write data port 604. Also during this cycle, the record for the sibling of this node, node 4, is read from RAM 610 using read address port 607 and read data port 605. In the example, this value of this record is 4.10 (ID=4, key=10). Read address 4 on line 607, is generated using the sibling unit 615 that complements the least-significant bit of current-node address 5 on line 606. Once the sibling record 4.10 has been read from RAM 610, it is held in sibling register 603. One skilled in the art will understand that if RAM 610 provides a static output, the value of the sibling record can be accessed directly from the RAM output port 605 and no sibling register is required. In either case, the key field of the sibling record 10 on line 608, is compared to the key field of the current record, 9 on line 609, by comparator 617. The output of the comparator 661 controls multiplexers 618 and 619 to select either the current record 5.9 from compare register 602 or the sibling record 4.10 from sibling register 603 to be output to the next stage on lines 613 and 614, for the ID and key fields respectively. In this case, 9 is less than 10, so the comparator selects 5.9 from the compare register to be output. In parallel with this selection, parent logic 616 computes the address of the node to be accessed in the next stage 10 by shifting the current address on line 606 right and setting the most significant bit. This address is output to the next stage on line 612.

A5

Please replace the paragraph at page 16, line 16 through line 24 with the following amended paragraph:

The pipelined scheduling engine of ~~Figure 8~~ Figures 8A-C computes a single incremental tournament in the same amount of time as the iterative engine of Figure 4. However, its advantage is that it can start a new incremental tournament each cycle. An example of this pipelined operation is illustrated in the table of Figure 9. The figure shows the contents of each register in the circuit on a different row as a function of time. Each timestep is shown in a separate column. The figure shows three incremental tournament computations: changing the counter of queue 5 to 9 (as shown in Figure 3), marking queue 7 disabled (as shown in Figure 7, in Figure 9 the disabled state is denoted by the letter d), and finally changing the counter of queue 2 to 15 (as shown in Figure 10).

A6

Please replace the paragraph at page 17, line 7 through line 14 with the following amended paragraph:

One skilled in the art will understand that the iterative engine of Figure 4 and the fully-pipelined engine of ~~Figure 8~~ Figures 8A-C are two extreme points on a continuum of design possibilities. With 4096 queues, for example, one could alternatively employ a six-stage pipeline where each stage handles two levels of the tree and performs two iterations before passing its result on to the next stage. Other combinations (e.g., four stages that perform three iterations each, three stages that perform four iterations each, or two stages that perform six iterations each) are also possible to trade off implementation complexity against performance.

A7

Please replace the paragraph at page 18, line 1 through line 6 with the following amended paragraph:

To allow the sorting network to handle a new set of updates each cycle, it can be pipelined in the same manner as the tournament search engine of ~~Figure 8~~ Figures 8A-C, with each stage of the sorting network completing its comparisons and exchanges in a given clock cycle. Also, to reduce hardware complexity, the sorting network may be updated in an incremental manner by using only a single comparison-exchange unit for each stage and operating it on only the input pairs that have changed in that stage.

A8

Please replace the paragraph at page 19, line 18 through line 28 with the following amended paragraph:

In the case where WSTT is less than or equal to the current time, transmission of the packet at the head of queue WCID is initiated in box 903. This packet is the CBR packet with the earliest STT and hence should be the first packet to be transmitted over the line. After packet transmission has been started, the STT for this queue is updated to reflect the bandwidth resources consumed by this packet transmission in box 904. The new STT for this packet is denoted NSTT. The details of this computation are shown in Figure 12 and described below. In box 905, the CBR tournament is updated to reflect the new STT for queue WCID. This update may be done using either the iterative circuit of Figure 4 or the pipelined circuit of ~~Figure 8~~ Figures 8A-C. Once the incremental tournament is complete and the transmission of the packet is complete, control returns to box 901 to decide which packet to send next.

A9

Please replace the paragraph at page 20, line 13 through line 18 with the following amended paragraph:

The same hardware can be used to perform both the WFQ tournaments and the CBR tournaments. During a given scheduling interval one tournament or the other is performed. There is never a need to perform both tournaments simultaneously. To support both tournaments in a single sorting engine, each RAM (200 in Figure 4 or 610, 630, and 650 in Figure 8C) is doubled in size to hold the trees for both tournaments and the high address bit is used to select the active tournament.

A10

Please replace the paragraph at page 23, line 10 through line 19 with the following amended paragraph:

The hardware engines of Figures 4 and 8A-C can be easily adapted to this alternative tournament search where only the result of each match is stored in the tree. The adaptation is performed by reducing the size of the RAMs to hold only the leaf nodes, adding a flip-flop array to hold the values of the internal nodes, and adding a set of multiplexers to compute the ID of match winners from the state of the flip flops. The RAM need not have a separate write port as it is written only at the beginning of the tournament. All subsequent state changes are recorded in

A11

A11 Cont.

the flip-flop array. In the pipelined implementation of ~~Figure 8~~ Figures 8A-8C, the individual RAMs in each pipeline stage must also be replaced by a single multi-ported RAM with a separate read port for each pipeline stage.

Please replace the paragraph at page 24, line 7 through line 15 with the following amended paragraph:

A12

The details of the flip-flop array 991 of Figure 19 are shown in ~~Figure 20~~ Figures 20A-B. The flip-flop array accepts the current node address on lines n3 to n0, and outputs the address of the winning leaf node for the sibling tournament on lines w2 to w0. Inverter 1010 converts the node address to a sibling address by complementing the LSB. Flip flops 958 to 964 hold the results of each comparison in the tournament. The flip flops are shown with exactly the same state as Figure 15. Multiplexers 1001 to 1007 compute the winner address based on the contents of the flip-flops and the input sibling address. Select boxes 1008 and 1009 are used to control multiplexers 1005 and 1006 to compute the bits of the winner address based on the stage of the tournament.

Please replace the paragraph at page 24, line 16 through line 25 with the following amended paragraph:

A13

The function of the flip-flop array of ~~Figure 20~~ Figures 20A-B is best understood by means of an example. Consider the case shown in Figure 15 where the previous winner, node 5, has been updated with a new value, 9, and the tournament must now be rerun, from the left to the right in Figure 15. To compute the winning address for the first match, the address of the sibling of node 5, 4, is applied to lines n3 to n0. Because line n3 is zero, the control input to all three of multiplexers 1005 to 1007 is zero and these multiplexers select their right most inputs passing node 4 to the output. Thus, the first match of the new tournament will be between leaf node 5, whose value is already in key register 312, and leaf node 4, whose value is read from the RAM. The result of this match, a 1 since node 5 still beats node 4, is used to update flip-flop 960.